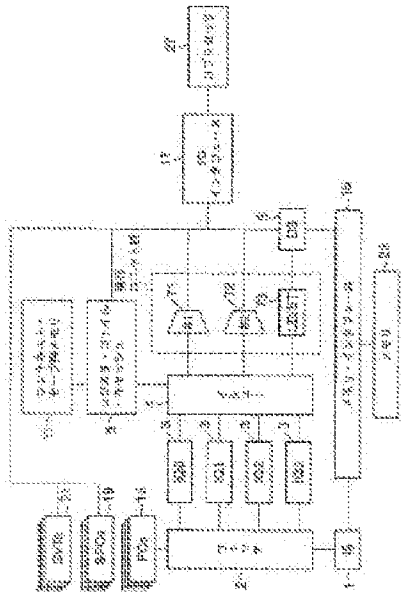
VIRTUAL MULTI-THREAD PROCESSOR AND THREAD EXECUTION METHOD

Publication number: JP2002163121 (A)
Publication date: 2002-06-07
Inventor(s): ASANO SHIGEHIRO; SAITO MITSUO
Applicant(s): TOKYO SHIBAURA ELECTRIC CO
Classification:
- international: ***G06F12/08; G06F9/46; G06F9/48;*** **G06F12/08; G06F9/46;** (IPC1-7): G06F9/46; G06F12/08
- European:
Application number: JP20000356238 20001122
Priority number(s): JP20000356238 20001122

Abstract of **JP 2002163121 (A)**

PROBLEM TO BE SOLVED: To provide a virtual multi-thread processor capable of handling many threads without being restricted by hardware. SOLUTION: A virtual thread number is allocated to a thread newly started on hardware threads 3 and 13 or the like. A register file cache 5 caches the contents of the registers of the threads, with the virtual thread identification numbers and register numbers used as keys. An issue section 4 performing issue control gains access to the register file cache 5 about the command to be issued, with the relevant thread identification number and the register identification number used as keys. When the cache is hit, it is determined that the command can be issued at least on the register. When the cache is missed, the relevant data are transferred between the register file cache 5 and a context saving memory 11.



(19)日本国特許庁（ＪＰ）　(12) **公開特許公報（Ａ）**　(11)特許出願公開番号
特開2002－163121
（P2002－163121A）
(43)公開日　平成14年６月７日(2002.6.7)

(51)Int.Cl.[7]	識別記号	ＦＩ		テーマコード（参考）
Ｇ０６Ｆ　9/46	３４０	Ｇ０６Ｆ　9/46	３４０Ｂ	５Ｂ００５
	３１３		３１３Ｄ	５Ｂ０９８
12/08	５５９	12/08	５５９Ｚ	
	５６５		５６５	

審査請求　未請求　請求項の数16　ＯＬ　（全 13 頁）

(21)出願番号　特願2000－356238(P2000－356238)

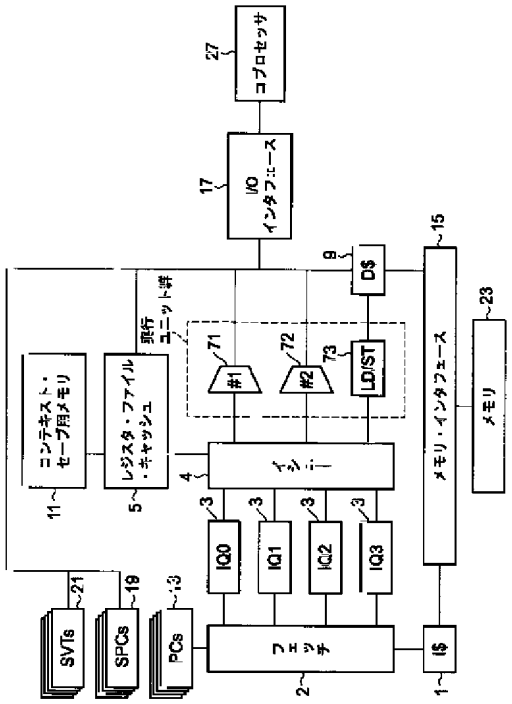
(22)出願日　平成12年11月22日(2000.11.22)

(71)出願人　000003078
株式会社東芝
東京都港区芝浦一丁目１番１号
(72)発明者　浅野　滋博
神奈川県川崎市幸区小向東芝町１番地　株式会社東芝研究開発センター内
(72)発明者　斎藤　光男
神奈川県川崎市幸区小向東芝町１番地　株式会社東芝マイクロエレクトロニクスセンター内
(74)代理人　100058479
弁理士　鈴江　武彦　（外６名）

最終頁に続く

(54)【発明の名称】　仮想マルチスレッドプロセッサ及びスレッド実行方法

(57)【要約】
【課題】　ハードウェアによる制限を受けずに多数のスレッドを扱うことのできる仮想マルチスレッドプロセッサを提供すること。
【解決手段】　ハードウェア・スレッド（３、１３等）上で新規に起動されたスレッドに、仮想スレッド番号を割り当てる。レジスタ・ファイル・キャッシュ５は、仮想スレッド識別番号およびレジスタ番号をキーとして各スレッドの各レジスタの内容をキャッシュする。発行制御を行うイシュー部４は、発行対象となる命令について該当するスレッド識別番号およびレジスタ識別番号をキーとしてレジスタ・ファイル・キャッシュ５をアクセスし、ヒットした場合に、少なくともレジスタに関しては該命令の発行は可能であると判断する。キャッシュ・ミスの場合には、レジスタ・ファイル・キャッシュ５とコンテキスト・セーブ用メモリ１１との間で該当するデータの転送が行われる。



【特許請求の範囲】
【請求項１】複数のスレッドを同時に実行できる仮想マルチスレッドプロセッサであって、
前記プロセッサ上で新規に起動されたスレッドに、該スレッドを特定する仮想スレッド識別番号を割り当てるための手段と、
各スレッドを特定する仮想スレッド識別番号および各レジスタを特定するレジスタ識別番号をキーとして各スレッドの各レジスタの内容をキャッシュするためのキャッシュ手段と、
前記プロセッサ上にある各スレッドの命令の発行制御を行う発行制御手段とを備え、
前記発行制御手段は、発行対象となる命令について該当する仮想スレッド識別番号およびレジスタ識別番号をキーとして前記キャッシュ手段をアクセスし、ヒットし且つ先行する命令の依存性が解決している場合に、少なくともレジスタに関しては該命令の発行は可能であると判断することを特徴とする仮想マルチスレッドプロセッサ。
【請求項２】前記発行制御手段は、発行可能な命令が複数あり且つ空いている演算ユニット数の制限のためにそれらのすべてを発行することができない場合には、所定の基準で選択された一部の命令のみ発行することを特徴とする請求項１に記載の仮想マルチスレッドプロセッサ。
【請求項３】前記発行制御手段により前記発行対象となる命令について前記仮想スレッド識別番号および前記レジスタ識別番号をキーとして前記キャッシュ手段がアクセスされ、キャッシュ・ミスが発生した場合に、該キャッシュ手段と所定のメモリとの間で該当するデータの転送を行うことを特徴とする請求項１に記載の仮想マルチスレッドプロセッサ。
【請求項４】前記プロセッサ上で実行中のスレッドがレーテンシの長いハードウェア資源に対して要求を出した場合に明示的に該スレッドを待避させる第１の処理と、待避中の前記スレッドが出した前記要求に対する前記所定のハードウェア資源からの応答があった後に該応答に対応する要求を出したスレッドを復帰すべきと判断される場合に該スレッドを復帰させる第２の処理とを行うための処理手段を更に備えたことを特徴とする請求項１に記載の仮想マルチスレッドプロセッサ。
【請求項５】前記処理手段は、前記第２の処理において、少なくとも一つのスレッドを起動するのに必要な資源の空きがあり、かつ、新規に起動すべき他のスレッドがない場合に、前記応答があったスレッドのいずれかを復帰すべきであると判断することを特徴とする請求項４に記載の仮想マルチスレッドプロセッサ。
【請求項６】前記仮想スレッド識別番号に対応して該仮想スレッド識別番号のスレッドが前記応答を返されて復帰可能な状態にあるかそれ以外の状態にあるかを少なくとも示す情報を保持する所定のレジスタを更に備え、
前記処理手段は、前記第２の処理において、前記所定のレジスタを参照することによって、復帰可能なスレッドを特定することを特徴とする請求項２に記載の仮想マルチスレッドプロセッサ。
【請求項７】前記処理手段における前記第２の処理の少なくとも一部を、特別なスレッドによって実行することを特徴とする請求項４に記載の仮想マルチスレッドプロセッサ。
【請求項８】前記スレッドからの前記所定のハードウェア資源への要求に、該スレッドの仮想スレッド識別番号を伴わせるとともに、該要求に対する該所定のハードウェア資源からの応答に、該要求を出した該スレッドの仮想スレッド識別番号を伴わせることによって、該所定のハードウェア資源からの応答と、該応答のもととなる要求を出したスレッドとを対応付けることを特徴とする請求項４に記載の仮想マルチスレッドプロセッサ。
【請求項９】前記所定のメモリは、前記プロセッサが命令によりアクセスするメモリとは別に専用のメモリとして構成されることを特徴とする請求項１に記載のマルチスレッドプロセッサ。
【請求項１０】複数のスレッドを同時に実行できる仮想マルチスレッドプロセッサにおけるスレッド実行方法であって、
前記プロセッサ上で新規に起動されたスレッドに、該スレッドを特定する仮想スレッド識別番号を割り当て、
各スレッドを特定する仮想スレッド識別番号および各レジスタを特定するレジスタ識別番号をキーとして各スレッドの各レジスタの内容をキャッシュ手段にキャッシュし、
前記プロセッサ上にある各スレッドの命令の発行制御を行う際に、発行対象となる命令について該当する仮想スレッド識別番号およびレジスタ識別番号をキーとして前記キャッシュ手段をアクセスし、ヒットし且つ先行する命令の依存性が解決している場合に、少なくともレジスタに関しては該命令の発行は可能であると判断することを特徴とするスレッド実行方法。
【請求項１１】複数のスレッドを同時に実行できる仮想マルチスレッドプロセッサであって、
前記プロセッサ上で新規に起動されたスレッドに、該スレッドを特定する仮想スレッド識別番号を割り当てるための手段と、
少なくとも一つの同期用カウンタと、
前記同期用カウンタの値を、同期をとるべきスレッドの数に設定するための手段とを備え、
同期をとるべき各々のスレッドにおいて同期をとるための同期命令に達した際には、前記同期用カウンタをデクリメントし、デクリメントされた後の前記同期用カウンタの値が０にならなければ、当該同期命令を発行したスレッドを復帰可能でない状態として待避させ、デクリメ

ントされた後の前記同期用カウンタの値が０になれば、当該同期命令を発行したスレッドは、先に同期命令に達し復帰可能でない状態として待避されたスレッドの状態を、復帰可能な状態に変更することを特徴とする仮想マルチスレッドプロセッサ。

【請求項１２】復帰可能な状態として待避されたスレッドを、所定の復帰条件が成立した場合に復帰させる処理を行うための手段を更に備えたことを特徴とする請求項１１に記載の仮想マルチスレッドプロセッサ。

【請求項１３】前記同期命令は前記同期用カウンタを識別する情報のフィールドを持つものであることを特徴とする請求項１１に記載のマルチスレッドプロセッサ。

【請求項１４】前記同期用カウンタを識別する情報として前記スレッドを識別する仮想スレッド識別番号を用いることを特徴とする請求項１３に記載のマルチスレッドプロセッサ。

【請求項１５】各スレッドを特定する仮想スレッド識別番号および各レジスタを特定するレジスタ識別番号をキーとして各スレッドの各レジスタの内容をキャッシュするためのキャッシュ手段を更に備え、

前記同期用カウンタも前記キャッシュ手段を通してアクセスできるようにしたことを特徴とする請求項１１に記載のマルチスレッドプロセッサ。

【請求項１６】複数のスレッドを同時に実行できる仮想マルチスレッドプロセッサにおけるスレッド実行方法であって、

前記プロセッサ上で新規に起動されたスレッドに、該スレッドを特定する仮想スレッド識別番号を割り当て、

同期用カウンタの値を、同期をとるべきスレッドの数に設定し、

同期をとるべき各々のスレッドにおいて同期をとるための同期命令に達した際には、前記同期用カウンタをデクリメントし、デクリメントされた後の前記同期用カウンタの値が０にならなければ、当該同期命令を発行したスレッドを復帰可能でない状態として待避させ、デクリメントされた後の前記同期用カウンタの値が０になれば、当該同期命令を発行したスレッドは、先に同期命令に達し復帰可能でない状態として待避されたスレッドの状態を、復帰可能な状態に変更することを特徴とすることを特徴とするスレッド実行方法。

【発明の詳細な説明】

【０００１】

【発明の属する技術分野】本発明は、複数のスレッドを実行可能な仮想マルチスレッドプロセッサ及びスレッド実行方法に関する。

【０００２】

【従来の技術】一つのプロセッサ上で複数のスレッドを実行するマルチスレッドプロセッサは、ＣＰＵ内部の演算器やメモリのレーテンシを隠蔽するための技術として広く知られている。

【０００３】しかしながら、従来のマルチスレッドプロセッサは、レジスタファイルなどのハードウェア資源をスレッドごとに用意することで複数のスレッドを実装するものであるので、実装できる（すなわちユーザが使用できる）スレッドの数は、レジスタファイルなどのハードウェアの資源の限界から限定されたものであった。

【０００４】したがって、一度に可能になるスレッドの数を例えば４程度に制限し、ハードウェア量が膨大になるのを防いでいた。これをプログラマの立場からみると、限定されたスレッドの数はプログラミングの自由度に対して大きな制約を課していることになり、問題があった。

【０００５】

【発明が解決しようとする課題】以上のように、従来のマルチスレッドプロセッサでは、スレッドの数がハードウェアの資源の限界から限定され、プログラミングの自由度に対して大きな制約を課していた。

【０００６】本発明は、上記事情を考慮してなされたもので、ハードウェアによる制限を受けずに多数のスレッドを扱うことのできる仮想マルチスレッドプロセッサを提供することを目的とする。

【０００７】また、マルチスレッドで記述されているプログラムでは、スレッド間の同期をとることが重要である。

【０００８】本発明は、効率的な同期手段を持つ仮想マルチスレッドプロセッサを提供することを目的とする。

【０００９】

【課題を解決するための手段】本発明は、複数のスレッドを同時に実行できる仮想マルチスレッドプロセッサであって、前記プロセッサ上で新規に起動されたスレッドに、該スレッドを特定する仮想スレッド識別番号を割り当てるための手段と、各スレッドを特定する仮想スレッド識別番号および各レジスタを特定するレジスタ識別番号をキーとして各スレッドの各レジスタの内容をキャッシュするためのキャッシュ手段と、前記プロセッサ上にある各スレッドの命令の発行制御を行う発行制御手段とを備え、前記発行制御手段は、発行対象となる命令について該当する仮想スレッド識別番号およびレジスタ識別番号をキーとして前記キャッシュ手段をアクセスし、ヒットし且つ先行する命令の依存性が解決している場合に、少なくともレジスタに関しては該命令の発行は可能であると判断することを特徴とする。

【００１０】好ましくは、前記発行制御手段は、発行可能な命令が複数あり且つ空いている演算ユニット数の制限のためにそれらのすべてを発行することができない場合には、所定の基準で選択された一部の命令のみ発行するようにしてもよい。好ましくは、前記発行制御手段により前記発行対象となる命令について前記仮想スレッド識別番号および前記レジスタ識別番号をキーとして前記キャッシュ手段がアクセスされ、キャッシュ・ミスが発

生した場合に、該キャッシュ手段と所定のメモリとの間で該当するデータの転送を行うようにしてもよい。好ましくは、前記プロセッサ上で実行中のスレッドがレーテンシの長いハードウェア資源に対して要求を出した場合に明示的に該スレッドを待避させる第１の処理と、待避中の前記スレッドが出した前記要求に対する前記所定のハードウェア資源からの応答があった後に該応答に対応する要求を出したスレッドを復帰すべきと判断される場合に該スレッドを復帰させる第２の処理とを行うための処理手段を更に備えるようにしてもよい。

【００１１】また、本発明は、複数のスレッドを同時に実行できる仮想マルチスレッドプロセッサであって、前記プロセッサ上で新規に起動されたスレッドに、該スレッドを特定する仮想スレッド識別番号を割り当てるための手段と、少なくとも一つの同期用カウンタと、前記同期用カウンタの値を、同期をとるべきスレッドの数に設定するための手段とを備え、同期をとるべき各々のスレッドにおいて同期をとるための同期命令に達した際には、前記同期用カウンタをデクリメントし、デクリメントされた後の前記同期用カウンタの値が０にならなければ、当該同期命令を発行したスレッドを復帰可能でない状態として待避させ、デクリメントされた後の前記同期用カウンタの値が０になれば、当該同期命令を発行したスレッドは、先に同期命令に達し復帰可能でない状態として待避されたスレッドの状態を、復帰可能な状態に変更することを特徴とする。

【００１２】好ましくは、復帰可能な状態として待避されたスレッドを、所定の復帰条件が成立した場合に復帰させる処理を行うための手段を更に備えるようにしてもよい。

【００１３】なお、装置に係る本発明は方法に係る発明としても成立し、方法に係る本発明は装置に係る発明としても成立する。

【００１４】本発明では、レジスタファイルを、仮想スレッド識別番号（ＶＴＩＤ）およびレジスタ識別番号（ＲＩＤ）をキーとして引かれるキャッシュとして構成する。プロセッサ上の資源に割り当てる仮想スレッド識別番号（ＶＴＩＤ）を適宜切り替えることによって、実際にハードウェア上に存在するスレッドより、仮想的に多くのスレッドが走行しているようにみせることができる。

【００１５】また、一般にプログラムによって使用されるレジスタの数は、用意されているレジスタの数より少ないことから、スレッドを切り替えるとき、レジスタファイルをすべてセーブ（Ｓａｖｅ）およびリストア（Ｒｅｓｔｏｒｅ）するかわりに、必要なものだけセーブおよびリストアすることができる。

【００１６】また、本発明では、複数のスレッドを同時に実行できるＳＭＴ（Ｓｉｍｕｌｔａｎｅｏｕｓ　Ｍｕｌｔｉ　Ｔｒｅａｄ）プロセッサにおいて、キャッシュされたレジスタファイルにミスしたときは、ミスしたレジスタを使用する命令の発行が行われないだけなので、プロセッサ全体がストールすることを回避できる。つまり、ＳＭＴプロセッサのイシュー・ロジックによって、レジスタファイルにミスしたスレッドは、レジスタが利用可能でない、すなわち発行可能でないと判定され、他のスレッドから発行可能な命令が発行されることになる。

【００１７】本発明によれば、スレッドの待避・復帰を可能とすることによって、プログラマにハードウェアで用意されたより多くのスレッドを見せることが可能になり、プログラミングの自由度が増す。また、その際に、マルチスレッドプロセッサ上のレジスタの内容などをメモリ上にセーブおよびリストアする回数を少なくし、実行時間を短縮することができる。

【００１８】また、本発明では、仮想スレッド番号（ＶＴＩＤ）ごとに同期機構を実現するためのカウンタを設ける。このカウンタは、同期にかかわる複数のスレッドが同期命令によってこのカウンタをデクリメントするように作用する。なお、カウンタの初期値は、同期にかかわるスレッドの数が設定されている。デクリメントの結果、カウンタの値が０でなかったスレッドは、Ｓｌｅｅｐ状態に入る。デクリメントの結果、カウンタが０であったスレッドは、同期処理ルーチンを実行する。同期処理ルーチンでは、同期命令の結果、Ｓｌｅｅｐ状態になっていたスレッドをＲｅａｄｙ状態にすることで、最後に同期命令を実行したスレッドと他のスレッドの同期が取れることになる。

【００１９】このように、本発明によれば、効率的にスレッド間の同期をとることができる。

【００２０】

【発明の実施の形態】以下、図面を参照しながら発明の実施の形態を説明する。

【００２１】（第１の実施形態）まず、本発明の第１の実施形態について説明する。

【００２２】図１に、本実施形態に係る仮想マルチスレッドプロセッサの構成例を示す。

【００２３】なお、図１では、本発明をＳＭＴ（Ｓｉｍｕｌｔａｎｅｏｕｓ　ＭｕｌｔｉＴｒｅａｄ）プロセッサに適用した例として、Ｉｎ－Ｏｒｄｅｒスーパースカラープロセッサの構成例を示している。

【００２４】まず、ハードウェア・スレッドの概念について説明しておく。ハードウェア・スレッドは、インストラクション・キュー（図１では３）に接続されている各プログラムカウンタ（図１では１３）に対応した概念で、図１の構成例の場合は４つのプログラムカウンタに対応して４つのハードウェア・スレッドがあることになる。ハードウェア・スレッドは、存在する複数のスレッドの中で、現にフェッチされているストリームを指している。本プロセッサで用意するハードウェア・スレッド

の数ｎは、基本的には、任意に設定（設計）可能である。ハードウェア・スレッドは、物理スレッド番号（ＰＴＩＤ（Ｐｈｙｓｉｃａｌ　ＴｈｒｅａｄＩＤ））で管理される。

【００２５】本実施形態では、詳しくは後述するように、本プロセッサにおいて各スレッドに固有の仮想スレッド番号（ＶＴＩＤ（Ｖｉｒｔｕａｌ　ＴｈｒｅａｄＩＤ））を割り当てて管理し、必要に応じてスレッドをハードウェア・スレッドからメモリへ待避させあるいはメモリからハードウェア・スレッドへ復帰させることによって、見かけ上、ハードウェア・スレッドの数ｎを上回る数のスレッドを扱うことができるようになっている。この意味におけるスレッドを、ハードウェア・スレッドに対して、仮想スレッドと呼ぶこととする。同時に扱うことのできる仮想スレッドの個数すなわち仮想スレッドに割り当てることのできる仮想スレッド番号（ＶＴＩＤ）の個数ｍも、基本的には、任意に設定（設計）可能である。例えば、仮想スレッド番号を８ビットで表現することにした場合には、最大２５６個の仮想スレッド番号が使用可能となる。

【００２６】さて、図１に示されるように、本プロセッサは、インストラクション・キャッシュ（Ｉｃａｃｈｅ）１、フェッチ部（Ｆｅｔｃｈ）２、ｎ組（本例では、ｎ＝４）のインストラクション・キュー（ＩＱ）３、ｎ組のプログラムカウンタ（ＰＣ）１３、スレッドの開始アドレスを保持するｎ組のＳＰＣレジスタ１９、仮想スレッド番号（ＶＴＩＤ）を保持するｎ組のＳＶＴレジスタ２１、イシュー部（Ｉｓｓｕｅ）４、レジスタ・ファイル・キャッシュ（Ｒｅｇｉｓｔｅｒ　Ｆｉｌｅｓ　Ｃａｃｈｅ）５、コンテキスト・セーブ用メモリ１１、複数の演算ユニット（本例では、２つの演算器７１，７２とロード・ストア・ユニット７３）、データ・キャッシュ（Ｄｃａｃｈｅ）９、メモリ・インタフェース（ＭｅｍｏｒｙＩ／Ｆ）１５、Ｉ／Ｏインタフェース（Ｉ／Ｏ　Ｉｎｔｅｒｆａｃｅ）１７を持つ。

【００２７】コンテキスト・セーブ用メモリ１１は、仮想スレッドに対応するデータ（例えば、対応するプログラムカウンタの内容、対応するステータスレジスタの内容、対応するレジスタファイルの内容など；以下、コンテキストと呼ぶ）を格納する。

【００２８】レジスタ・ファイル・キャッシュ５は、コンテキスト・セーブ用メモリ１１に格納されている仮想スレッドのコンテキストをキャッシュする。

【００２９】なお、レジスタ・ファイルをキャッシュする技術としては、例えば、文献１(Peter R.Nuth, William J.Dally, “The Named-State Register File:Implementation and Performance”, First Ann. International Symposium High-Performance Computer Architecture,pp.4-13,Jan.1995)に開示されている技術を用いることができる。

【００３０】また、メモリ・インタフェース１５に、メモリ（Ｍｅｍｏｒｙ）２３が接続される。

【００３１】また、Ｉ／Ｏインタフェース１７に、例えばコプロセッサ（Ｃｏｐｒｏｃｅｓｓｏｒ）２７などが接続される。

【００３２】図１に示されるように、本仮想マルチスレッドプロセッサでは、個々のハードウェア・スレッド専用に、プログラムカウンタ（ＰＣ）およびインストラクション・キュー（ＩＱ）などの資源が設けられる。４つのハードウェア・スレッドをサポートする場合、４組のプログラムカウンタ（ＰＣ）およびインストラクション・キュー（ＩＱ）などが必要になる。そして、ハードウェア・スレッドへの仮想スレッドの対応をスレッドの実行に応じて適宜切り替えるようにしている。なお、本プロセッサでは、複数のスレッドで演算ユニットやキャッシュなどは共有する構成にしている。

【００３３】なお、図１においては接続線の一部を省略してある。

【００３４】ここで、本プロセッサの一般的な動作について説明する。

【００３５】インストラクション・キャッシュ１は、メモリ２３から命令をキャッシュする。インストラクション・キャッシュ１がミスしたときは、メモリ２３からインストラクション・キャッシュ１へ命令がリフィル（Ｒｅｆｉｌ）される。

【００３６】データ・キャッシュ９は、メモリ２３からデータをキャッシュする。データ・キャッシュ９がミスしたときは、メモリ２３からデータ・キャッシュ９へデータがリフィルされる。

【００３７】フェッチ部２は、命令のフェッチ（インストラクションキャッシュ１へのアクセス）を行う。

【００３８】インストラクション・キュー３は、各スレッドに対応する命令を保持する。

【００３９】イシュー部４は、命令の発行制御と、命令のデコードやレジスタファイルへのアクセスを行う。

【００４０】演算器７１，７２やロード・ストア・ユニット７３は、発行された命令を実行する。その際、必要に応じて所定のタイミングで、データ・キャッシュ９やレジスタファイル５に対するアクセス（リード、あるいはライト）などを行う。

【００４１】なお、このプロセッサは、パイプライン・プロセッサであってもよい。例えば、５段のパイプラインの場合、次のようなステージ構成になる。

フェッチ・ステージ…命令のフェッチ

↓

デコード・ステージ…命令のデコード、レジスタファイルへのアクセス

↓

実行ステージ…命令の実行

↓

メモリ・ステージ…メモリへのアクセス
↓
ライトバック(Writeback)ステージ…レジスタファイルへの書き込み
次に、仮想スレッドの起動に関して説明する。
【００４２】まず、本実施形態では、スレッドの起動には、次の２つがある。一つは、スレッドが新規に起動される場合で、これを新規起動と呼ぶものとする。もう一つは、新規起動された後に一旦メモリに待避されたスレッド（後述するＳｌｅｅｐ状態からＲｅａｄｙ状態になったスレッド）が復帰されて起動する場合で、これを復帰起動と呼ぶものとする。
【００４３】スレッドの新規起動は、ハードウェア上で各スレッドが専有する必要のある資源（プログラムカウンタおよびインストラクション・キュー等）のうち空いているものが少なくとも１スレッド分存在する場合に、行うことができる（ここでは、この場合に対象スレッドがあればそれを優先的に新規起動させるものとする）。例えば、図１において、ハードウェア上で３つのスレッドが走行中の場合、４組のプログラムカウンタおよびインストラクション・キューなどのうち３組が使用されており、１スレッド分は使用されていないので、１つのスレッドを新規起動することができる。
【００４４】スレッドの新規起動は、使用するハードウェア・スレッド（例えば、ハードウェア・スレッド番号ＰＴＩＤ＝０、１、２または３）に専用に設けられるＳＰＣレジスタ１９（例えば、ＳＰＣ０、ＳＰＣ１、ＳＰＣ２またはＳＰＣ３）に、当該スレッドの開始アドレスを設定しておき（なお、該開始アドレスはユーザが記述した命令の実行によって設定される）、起動命令により、該ＳＰＣレジスタ１９から対応するプログラムカウンタ１３（例えば、ＰＣ０、ＰＣ１、ＰＣ２またはＰＣ３）に値を転送することによって、開始される。このとき、該ハードウェア・スレッドに専用に設けられるＳＶＴレジスタ２１（例えば、ＳＶＴ０、ＳＶＴ１、ＳＶＴ２またはＳＶＴ３）には、仮想スレッド番号を設定しておき、新規起動されたスレッドは、それが起動されたハードウェア・スレッドに対応するＳＶＴレジスタ２１に設定された仮想スレッド番号を持つようになる。
【００４５】なお、ここでは、上記の仮想スレッド番号の割り当ておよび設定は、ソフトウェアによって（ユーザが記述した命令の実行によって）行うことにしているが、ハードウェアによって実現することも可能である。ハードウェアによって仮想スレッド番号の割り当てを自動的に行う場合には、例えば、各仮想スレッド番号について使用中か未使用であるかを知るための情報（例えば、仮想スレッド番号と使用／未使用フラグとのテーブル、使用していない仮想スレッド番号のプール）を管理し、スレッドが新規起動する際には未使用の仮想スレッド番号から所定の基準（例えば、ランダム、番号の小さい順等）で選択したものを割り当ててその仮想スレッド番号を使用中として管理し、スレッドが終了する際にはそのスレッドが使用していた仮想スレッド番号を未使用として管理するようにすればよい。
【００４６】次に、イシュー部４の処理について説明する。
【００４７】図１において、各スレッドの各命令は、メモリ２３から、メモリ・インタフェース１５／インストラクション・キャッシュ１／フェッチ部２を経由して、スレッド対応のインストラクション・キュー３に入る。図１の例では、インストラクション・キュー３は、ハードウェア上に存在する４つのスレッドに対応して４つ（ＩＱ０、ＩＱ１、ＩＱ２、ＩＱ３）ある。
【００４８】イシュー部４は、所定のイシュー・ロジックに従って、各スレッドに対応する各インストラクション・キュー３の先頭の命令の各々について、それが実行可能か否か判断し、実行可能な命令があれば、該命令（図１の例では、最大３つ）を、該当する実行ユニット（図１の例では、７１，７２，７３のいずれか）に送る。例えば、インストラクション・キューＩＱ０の命令を、ロード・ストア・ユニット７３に送り、インストラクション・キューＩＱ１の命令を、演算器７１に送り、インストラクション・キューＩＱ３の命令を、演算器７２に送る。
【００４９】所定のイシュー・ロジックすなわち実行可能な命令を判定するロジックとしては、例えば、その時点で、当該命令に対応する実行ユニットが空いていて、かつ、レジスタ・ファイル・キャッシュ５に対応するレジスタが存在しており、かつ、対応するレジスタが先行する命令で書き込みに指定されている場合は、その書き込みが終了した場合に、実行可能と判断するものである。また、その際に、１以上の実行ユニットが空いており、空いている実行ユニットを使うことができれば実行可能となる命令が複数ある場合に、それら命令のすべてを実行するには、空いている実行ユニットの数が不足するときは、それら命令を先頭に持つ複数のインストラクション・キュー３のうちから所定の選択基準に従って一部のものが選択され、選択されたインストラクション・キュー３の先頭の命令が、空いている実行ユニットに割り当てられる。
【００５０】所定の選択基準としては、ランダムに選択する方法の他に、インストラクション・キューに着目する方法として、その時点でのキュー長が短いインストラクション・キュー３を優先的に選択する方法、予め各インストラクション・キュー３に付与された固定の優先順位に基づいて選択する方法など、様々な方法が考えられる。あるいは、スレッドに着目する方法として、仮想スレッド番号が小さいスレッド（に対応するインストラクション・キュー）から選択する方法、各スレッドについて（例えばユーザが命令を記述することにより）優先度

を設定し、より優先度の高いもの（に対応するインストラクション・キュー）から選択する方法など、種々の方法がある。また、それらを適宜組み合わせた方法も可能である。

【００５１】例えば、演算器７１とロード・ストア・ユニット７３が空いており、インストラクション・キューＩＱ０の命令は、ロード・ストア・ユニット７３で実行可能であり且つレジスタ・ファイル・キャッシュ５に対応するレジスタが存在しており、インストラクション・キューＩＱ１～ＩＱ４の命令は、演算器７１で実行可能であり且つレジスタ・ファイル・キャッシュ５に対応するレジスタが存在しているものとする。ロード・ストア・ユニット７３で実行可能な命令はインストラクション・キューＩＱ０の命令だけであるので、インストラクション・キューＩＱ１の命令を、演算器７１に送ればよい。一方、演算器７１で実行可能な命令はインストラクション・キューＩＱ１～ＩＱ４の３つの命令であるので、発行すべき命令（のインストラクション・キュー）の選択が必要となり、例えばインストラクション・キューＩＱ１が選択されたとすると、その命令を、演算器７２に送ることになる。

【００５２】次に、レジスタ・ファイル・キャッシュ５の働きについて説明する。

【００５３】レジスタ・ファイル・キャッシュ５は、スレッドに付けられた仮想スレッド番号（ＶＴＩＤ）とレジスタ番号（ＲＩＤ）とをキーとして引かれるキャッシュになっている。レジスタ・ファイル・キャッシュ５について、キャッシュにミスした場合は、コンテキスト・セーブ用メモリ１１から必要なデータがフェッチされる。また、キャッシュの容量には、ハードウェア上の制約があるので、ＬＲＵなどのアルゴリズムなどによりデアロケートされるエントリが決定され、コンテキスト・セーブ用メモリ１１に書き込まれる。

【００５４】なお、レジスタファイルキャッシュ５について、キャッシュ・ミスは、イシュー部４が、各スレッドに対応する各インストラクション・キュー３の先頭の命令の各々について、それが実行可能か否か判断する際に、レジスタ・ファイル・キャッシュ５に対応するレジスタが存在しているか否かを調べた際に発生する。この場合は、コンテキスト・セーブ用メモリ１１から必要なデータがフェッチされるので、イシュー部４が当該命令について次に判断を行う際には、レジスタ・ファイル・キャッシュ５には対応するレジスタが存在している状態になっている。

【００５５】一方、実行ユニット（例えば７１等）の演算結果やメモリ２３からデータ・キャッシュ９経由で読み出されたデータは、レジスタ・ファイル・キャッシュ５に書き戻される。

【００５６】なお、デアロケート時にコンテキスト・セーブ用メモリ１１に書き込まなくてもよい場合を検出するため、ソフトウェアからｄｅａｄなレジスタを示す手法を用いてもよい。この手法によれば、ｄｅａｄなレジスタは、コンテキスト・セーブ用メモリ１１に書き込む必要がなく、新たなレジスタが必要になった場合のアロケーションの速度を早くすることができる。この手法としては 例えば 文献2(Jack L.Lo et.al, “Software-Directed Register Deallocation for Simultaneous Multithreaded Processors”,IEEE Trans. on Parallel and Distributed Systems, Vol.10,No.9,Sep,1999)に開示された技術を用いることができる 次に、仮想スレッドの状態およびその遷移について説明する。

【００５７】図２に、仮想スレッドの４つの状態と、状態間の遷移関係を示す。

【００５８】仮想スレッドの状態には、パイプライン上を流れている状態（Ａｃｔｉｖｅ状態）、インストラクション・キュー３にはあるが実行できない状態（Ｗａｉｔ状態）、同期など応答を待っている状態（Ｓｌｅｅｐ状態）、Ｓｌｅｅｐ状態から応答が返って実行可能な状態（Ｒｅａｄｙ状態）の４つがある。

【００５９】状態間の遷移としては、Ａｃｔｉｖｅ状とＷａｉｔ状態との間の遷移関係と、Ａｃｔｉｖｅ状態からＳｌｅｅｐ状態へ／Ｓｌｅｅｐ状態からＲｅａｄｙ状態へ／Ｒｅａｄｙ状態からＡｃｔｉｖｅ状態への遷移関係との２系統がある。

【００６０】Ａｃｔｉｖｅ状からＷａｉｔ状態への遷移は、キャッシュ・ミスやレーテンシなどによって起こり、それが解消することによって、Ａｃｔｉｖｅ状からＷａｉｔ状態へ戻る。

【００６１】Ａｃｔｉｖｅ状態からＳｌｅｅｐ状態への遷移は、例えば命令のレーテンシが長い場合、例えばコプロセッサ命令の場合（あるいは他の理由でスリープすべき場合）などに、明示的に行われる。

【００６２】Ｓｌｅｅｐ状態からＲｅａｄｙ状態への遷移は、Ｓｌｅｅｐ状態に遷移するときに出した命令あるいは要求（例えばあるコプロセッサにある処理の実行を要求する命令）に対応する応答が、（例えば要求を受けたコプロセッサから）返されたことを契機として行われる。なお、この他に、あるスレッドがＳｌｅｅｐ状態の他のスレッドをＲｅａｄｙ状態にすることを指示する命令を用意することも可能である。

【００６３】Ａｃｔｉｖｅ状態のスレッドがＳｌｅｅｐ状態に遷移すると、いままでインストラクション・キュー３に命令を詰めていた当該スレッドに対応するプログラムカウンタ１３は無効化される（インストラクション・キュー３内の命令も無効化される）。なお、この時点で、レジスタ・ファイル・キャッシュ５の該当するデータを維持しておく方法と、コンテキスト・セーブ用メモリ１１に追い出す方法とがある。

【００６４】次に、Ｒｅａｄｙ状態からＡｃｔｉｖｅ状態への遷移について説明する。

【００６５】上記のように、Ａｃｔｉｖｅ状態のスレッドがＳｌｅｅｐ状態に遷移して、対応するプログラムカウンタ１３は無効化されると、新規起動または復帰起動が可能になる。ここでは、新規起動できるスレッドがあれば新規起動を行い、新規起動できるスレッドがなく且つ復帰起動できるスレッドがあれば復帰起動するものとする。なお、新規起動できるスレッドも復帰起動できるスレッドもなければ、新規起動できるスレッドまたは復帰起動できるスレッドが出現するまで、当該ハード・ウェアスレッド資源は未使用（空き状態）になる。

【００６６】さて、Ｒｅａｄｙ状態のスレッドを復帰起動できることとなった場合に、本実施形態では、システムタスクと呼ばれる特殊なスレッドによって、Ｒｅａｄｙ状態のスレッドが複数ある場合におけるスレッドの選択と、スレッドの復帰起動のための処理を行うものとする。このメカニズムにより、実行中のスレッドの邪魔をしないことを可能にしている。

【００６７】この場合、前述のようにして無効化されたプログラムカウンタ１３には、新たにシステムタスクの番地がセットされる。

【００６８】起動されたシステムタスクは、Ｒｅａｄｙ状態になっているスレッドのうちから１つを選択し、そのスレッドのプログラムカウンタの値を、対応するプログラムカウンタ１３にセットするなどの処理を行う。すると、該プログラムカウンタの値に基づいて、命令が新たにフェッチされ始め、対応するインストラクション・キュー３に、命令が入れられる。

【００６９】なお、Ｒｅａｄｙ状態になっているスレッドが複数ある場合に、システムタスクが、それらから１つを選択するアルゴリズムには、様々なものが考えられる。例えば、ランダムに選択する方法、ＦＩＦＯでもっとも古くＲｅａｄｙ状態になったものを選択する方法、仮想スレッド番号が小さいものから選択する方法、各スレッドについて（例えばユーザが命令を記述することにより）優先度を設定し、より優先度の高いものを選択する方法など、種々の方法がある。

【００７０】なお、あるスレッドがＲｅａｄｙ状態の他のスレッドをＡｃｔｉｖｅ状態にすることを指示する命令を用意することも可能である。

【００７１】ここで、Ｒｅａｄｙ状態のスレッドを高速に検索するためには、例えば、仮想スレッド番号に対応して、Ｒｅａｄｙ状態の場合にＲｅａｄｙビットに１が立つ（それ以外の場合にはＲｅａｄｙビットは０とする）ようにしたＲｅａｄｙレジスタを設ける方法がある（なお、Ｓｌｅｅｐ状態（例えば、０１）か、Ｒｅａｄｙ状態（例えば、１０）か、またはそれ以外の状態（例えば、００）かを示すようにする方法もある）。システムタスクでは、Ｒｅａｄｙレジスタを参照し、Ｒｅａｄｙビットに１（あるいは、１０）が立っているスレッドを探索し、それが１つあればそのスレッドを、複数あればそれらから選択した１つのスレッドを、ハードウェア・スレッドにする。

【００７２】なお、本実施形態では、要求（例えばコプロセッサ命令）およびその応答には、仮想スレッド番号を付加するものとする。すなわち、要求を出す際には、その要求元のスレッドの仮想スレッド番号を該要求に付加するものとする。応答については、例えば、要求を受けた資源が応答を出す際に、該要求に付加されていた仮想スレッド番号を該応答に付加する。また、例えば、要求／応答と仮想スレッド番号との対応を管理する管理部を設け、要求を出す際には、その要求元のスレッドの仮想スレッド番号を該要求に付加して一旦管理部に渡し、管理部が仮想スレッド番号の付加されていない要求を資源に渡し、該資源は仮想スレッド番号の付加されていない応答を返し、管理部は、この応答を受け、対応する仮想スレッド番号を特定する、という構成をとることも可能である。このようにすれば、応答が返ってきた場合（例えば、コプロセッサ命令の実行などでＳｌｅｅｐ状態になっているスレッドへコプロセッサ命令などから応答が返ってきた場合）は、応答が仮想スレッド番号（ＶＴＩＤ）とともに返ってくるので、この仮想スレッド番号によってＲｅａｄｙレジスタの対応するビットを立てるだけでよい。

【００７３】なお、上記では、仮想スレッド番号に対応してＲｅａｄｙビットを持つＲｅａｄｙレジスタを設ける方法を示したが、Ｓｌｅｅｐ状態またはＲｅａｄｙ状態のスレッドの仮想スレッド番号とその状態を示す情報（例えば、Ｓｌｅｅｐ状態のとき０、Ｒｅａｄｙ状態のとき１）とを１つのエントリとして含むＲｅａｄｙテーブルを設けることも可能である。

【００７４】ここで、ＲｅａｄｙレジスタのＲｅａｄｙビットが、Ｓｌｅｅｐ状態か、Ｒｅａｄｙ状態か、またはそれ以外の状態（ｎｕｌｌ状態と呼ぶ）（例えば、起動されたが待避中でない状態、あるいは、まだ新規に起動されていない状態（スレッドが終了して該仮想スレッド番号が解放された状態を含む））をとる場合（あるいはＲｅａｄｙレジスタ（あるいは、ＲｅａｄｙテーブルのＲｅａｄｙビットが、Ｓｌｅｅｐ状態か、またはＲｅａｄｙ状態をとる場合）の処理手順について説明する。

【００７５】図3に、Ｓｌｅｅｐ状態に遷移する場合として、コプロセッサ命令などのレーテンシの長い命令が実行される際の処理手順の一例を示す。

【００７６】ある仮想スレッド番号のスレッドについてコプロセッサ命令などのレーテンシの長い命令が実行されたときは、Ｒｅａｄｙレジスタ（あるいは、Ｒｅａｄｙテーブル）の当該仮想スレッド番号に対応するＲｅａｄｙビットを、Ｓｌｅｅｐ状態を示す値に設定し（ステップＳ１１）、該当するハードウェア・スレッドを無効化する（ステップＳ１２）。なお、ステップＳ１１とステップＳ１２は、上記とは逆の順番で行ってもよいし、

並列的に行ってもよい。

【００７７】図４に、応答を受けた場合の処理手順の一例を示す。

【００７８】応答があった場合、応答に伴って返される仮想スレッド番号から、Ｒｅａｄｙレジスタ（あるいは、Ｒｅａｄｙテーブル）を検索する（ステップＳ２１）。Ｒｅａｄｙレジスタ（あるいは、Ｒｅａｄｙテーブル）において、応答に伴って返された仮想スレッド番号と同じ仮想スレッド番号を持つスレッドであってそのＲｅａｄｙビットがＳｌｅｅｐ状態を示すものがあれば（ステップＳ２２）、該ＲｅａｄｙビットをＲｅａｄｙ状態を示す値に変更する（ステップＳ２３）。

【００７９】なお、応答に伴って返された仮想スレッド番号がいずれのテーブルにも登録されていない場合には、何もしないこととするか、または所定のエラー処理をすることとする（ステップＳ２４）。

【００８０】なお、応答があってから、対応するスレッドが復帰するまでの間、応答（に関するデータ；例えば、戻り値などを含む）は、仮想スレッド番号に対応付けて保持される。

【００８１】図５に、システムタスクに関する処理手順の一例を示す。

【００８２】Ｒｅａｄｙレジスタ（あるいは、Ｒｅａｄｙテーブル）にＲｅａｄｙビットがＲｅａｄｙ状態を示すスレッドがある場合に、ハードウェア・スレッドに空きがあり、かつ、新規起動すべきスレッドがなければ、当該空きハードウェア・スレッドでシステムタスクが起動される（ステップＳ３１）。起動されたシステムタスクは、Ｒｅａｄｙ状態のスレッドが１つならばそのスレッドを復帰させるものとし、Ｒｅａｄｙ状態のスレッドが複数あるならば復帰させるものとして１つのスレッドを選択し、その復帰させるスレッドの仮想スレッド番号（ＶＴＩＤ）に基づいて、プログラムカウンタの値の復帰などの処理を行い、復帰起動させる（ステップＳ３２）。なお、Ｒｅａｄｙレジスタでは、復帰されたスレッドに対応するＲｅａｄｙビットがｎｕｌｌ状態にされ、Ｒｅａｄｙテーブルでは復帰されたスレッドに対応するエントリが無効にされる。

【００８３】これら仮想スレッドに関する処理は、イシュー部４が行うようにしてもよいし、イシュー部４以外の制御部を設けてもよい。

【００８４】以上説明したように、本発明によれば、スレッドの待避・復帰を可能とすることによって、プログラマにハードウェアで用意されたより多くのスレッドを見せることが可能になり、プログラミングの自由度が増す。また、その際のスレッドの切り替えも、効率且つ高速に行うことができる。

【００８５】また、上記では、システムタスクをスレッドで実現したが、システムタスクを専用のハードウェアで構成することも可能である。

【００８６】また、これまでの説明では、Ｒｅａｄｙ状態のスレッドは、ハードウェア・スレッドに空きが存在し、かつ、新規起動すべきスレッドがない場合に、復帰起動することができるものとしたが、新規起動すべきスレッドがあっても、Ｒｅａｄｙ状態のスレッドを復帰起動することを可能としてもよい。例えば、ハードウェア・スレッドに空きが存在するときに、１または複数の新規起動すべきスレッドと、１または複数のＲｅａｄｙ状態のスレッドがあった場合に、予め定められた選択基準に基づいて選択した１つの新規起動すべきスレッドまたはＲｅａｄｙ状態のスレッドを新規起動または復帰起動させるようにしてもよい。例えば、各スレッドについて（例えばユーザが命令を記述することにより）優先度を設定し、より優先度の高いものを選択する方法、新規起動すべきスレッドの数とＲｅａｄｙ状態のスレッドの数とに基づいて選択する方法、Ｒｅａｄｙ状態のスレッドの数が所定数を超えた場合にのみＲｅａｄｙ状態のスレッドを優先して選択する方法など、種々の方法がある。

【００８７】また、これまでの説明は、ＳＭＴ（Ｓｉｍｕｌｔａｎｅｏｕｓ　ＭｕｌｔｉＴｒｅａｄ）プロセッサの例として、Ｉｎ－Ｏｒｄｅｒスーパースカラープロセッサを中心として行ったが、本発明はＯｕｔ－ｏｆ－Ｏｒｄｅｒプロセッサにも適用可能である。

【００８８】（第２の実施形態）次に、本発明の第２の実施形態に係るスレッド間の同期方法について説明する。

【００８９】以下で説明するスレッド間の同期方法は、実行時に資源の割り当てをダイナミックに行う第１の実施形態の仮想マルチプロセッサ（または、第１の実施形態の仮想マルチプロセッサの構成のうち本実施形態にとって必要な部分のみ備える仮想マルチプロセッサ）に適用した場合を例にとって説明するが、本発明は、その他、第１の実施形態をＯｕｔ－ｏｆ－Ｏｒｄｅｒにした場合の仮想マルチプロセッサ、シングルイシュー（Ｉｓｓｕｅ）の仮想マルチプロセッサ、コンパイル時にスタティックに資源を割り当て使用するＶＬＩＷ（Ｖｅｒｙ　Ｌｏｎｇ　Ｉｎｓｔｒｕｃｔｉｏｎ　Ｗｏｒｄ）の仮想マルチプロセッサ、それらにおいてレジスタ・ファイルをハードウェア・スレッド毎に持つようにしたものなど、種々の仮想マルチプロセッサに対しても適用可能である。

【００９０】さて、マルチスレッドのプログラムにおいては、複数のプログラムが終了したことを互いに知ることが必要となる。例えば、並列に走行しているスレッドＡとスレッドＢとスレッドＣの結果をスレッドＤが使う場合に、スレッドＡとスレッドＢとスレッドＣは同期を取らなければならない。本実施形態では、このような場合に、ＳＹＮＣ命令（同期命令）を使うものとする。ＳＹＮＣ命令には、カウンタを操作する機能と、同期をとるための機能とが含まれる。

【００９１】本実施形態では、仮想スレッド番号（ＶＴＩＤ）ごとに同期機構を実現するためのカウンタを設けるものとする。

【００９２】今、スレッドＡの仮想スレッド番号（ＶＴＩＤ）を０、スレッドＢの仮想スレッド番号（ＶＴＩＤ）を１、スレッドＣの仮想スレッド番号（ＶＴＩＤ）を２とし、同期のためのカウンタとして、スレッドＡのＶＴＩＤ＝０に対応するカウンタを使用するものとする（いずれのスレッドに対応するカウンタを使用するかは、ユーザが任意に決めて構わない）。このカウンタをＳＹＮＣカウンタ（同期カウンタ）と呼ぶ。ＳＹＮＣカウンタの初期値としては、スレッドＡが、ＳＹＮＣを行うスレッドの数３に対応して、３を設定しておく（例えば、ユーザが、同期をとるスレッドの数を設定する命令を記述する）。

【００９３】まず、最初の例では、スレッドＡが最後にＳＹＮＣ命令に到達した場合を示す。

【００９４】図６で示すように、最初にＳＹＮＣ命令に到達したスレッドＣは、ＳＹＮＣカウンタをデクリメントし、その値を２とする。ＳＹＮＣカウンタの値が０でないので、このスレッドは、Ｓｌｅｅｐ状態に遷移する。

【００９５】次にＳＹＮＣ命令に到達したスレッドＢは、ＳＹＮＣカウンタをデクリメントして、その値を１とする。まだ、ＳＹＮＣカウンタは０でないので、スレッドＣもＳｌｅｅｐ状態に遷移する。

【００９６】最後にＳＹＮＣ命令に到達したスレッドＡでは、ＳＹＮＣカウンタをデクリメントし、その結果が０であるので、他のスレッドを起こすためのルーチンを実行する。このルーチンでは、スレッドＢおよびスレッドＣのＲｅａｄｙレジスタ（あるいは、Ｒｅａｄｙテーブル）のＲｅａｄｙビットを１に立てることで、Ｓｌｅｅｐ状態からＲｅａｄｙ状態へ遷移させる。

【００９７】Ｒｅａｄｙ状態に遷移した後は、例えば、（条件が整えば）システムスレッドによってハードウェア・スレッドに復帰することができる。

【００９８】ＳＹＮＣ命令は、次の２つの情報のフィールドを持つものとする。

【００９９】一つの情報は、ＳＹＮＣカウンタを識別する識別子で、これには仮想スレッド番号（ＶＴＩＤ）を使えばよい。なお、複数のスレッドが走行し、複数の同期ポイントを同時に設定する可能性があることを想定する場合には、ＳＹＮＣカウンタを複数設ける。

【０１００】もう一つの情報は、ＳＹＮＣ命令を実行した後、ＳＹＮＣカウンタが０になった場合に実行するルーチンの先頭番地である。上記の例では、例えば、スレッドＡとスレッドＢとスレッドＣのうちＳｌｅｅｐ状態のものをＲｅａｄｙ状態にするルーチンの先頭番地である。

【０１０１】ＳＹＮＣカウンタは、必ずしもハードウェア上に常に存在する必要はなく、先に説明したレジスタ・ファイル・キャッシュ５に入れられてもよい。すなわち、ＳＹＮＣカウンタは、仮想スレッド番号ごとに存在しているので、仮想スレッド番号とＳＹＮＣカウンタを示すレジスタ番号とをキーとすれば、通常のレジスタと全く同じ仕組みでアクセスが可能となり、コンテキスト・セーブ用メモリ１１に必要に応じてセーブもしくはリストアされる。

【０１０２】図７に、スレッドがＳＹＮＣ命令に到達した場合の処理手順の一例を示す。

【０１０３】ＳＹＮＣ命令に到達したならば、まず、ＳＹＮＣカウンタをデクリメントする（ステップＳ４１）。

【０１０４】ここで、デクリメントした後のＳＹＮＣカウンタの値が０でなければ（ステップＳ４２）、当該スレッドを、Ｓｌｅｅｐ状態に遷移させる（ステップＳ４４）。

【０１０５】一方、デクリメントした後のＳＹＮＣカウンタの値が０であれば（ステップＳ４２）、同期をとる関係にある全スレッドのうちＳｌｅｅｐ状態にあるものをＲｅａｄｙ状態にするためのルーチンを実行する（ステップＳ４３）。Ｒｅａｄｙ状態に遷移した後は、例えばシステムスレッドによって復帰される。

【０１０６】なお、これらスレッド間の同期に関する処理は、例えば図１の構成の場合には、イシュー部４が行うようにしてもよいし、イシュー部４以外の制御部を設けてもよい。

【０１０７】本発明を用いないビジーウェイトでは、ＣＰＵが無駄に浪費されるのみならず、ビジーウェイトの箇所がハードウェアスレッドの数に等しいか多ければデッドロックが起こる。また、割り込みを用いる方法では、オーバーヘッドが大きい。これに対して、本発明のマルチスレッド間の同期方法を用いれば、それら問題を回避して、効果的な同期を行うことができる。

【０１０８】なお、本実施形態で例示した構成は一例であって、それ以外の構成を排除する趣旨のものではなく、例示した構成の一部を他のもので置き換えたり、例示した構成の一部を省いたり、例示した構成に別の機能を付加したり、それらを組み合わせたりすることなどによって得られる別の構成も可能である。また、例示した構成と論理的に等価な別の構成、例示した構成と論理的に等価な部分を含む別の構成、例示した構成の要部と論理的に等価な別の構成なども可能である。また、例示した構成と同一もしくは類似の目的を達成する別の構成、例示した構成と同一もしくは類似の効果を奏する別の構成なども可能である。また、各種構成部分についての各種バリエーションは、適宜組み合わせて実施することが可能である。また、本実施形態は、装置としての発明、装置内部の構成部分についての発明、またはそれらに対応する方法の発明等、種々の観点、段階、概念またはカ

テゴリに係る発明を包含・内在するものである。従って、この発明の実施の形態に開示した内容からは、例示した構成に限定されることなく発明を抽出することができるものである。

【０１０９】本発明は、上述した実施の形態に限定されるものではなく、その技術的範囲において種々変形して実施することができる。

【０１１０】

【発明の効果】本発明によれば、レジスタ・ファイルを、仮想スレッド識別番号およびレジスタ識別番号をキーとするキャッシュとして構成することで、プロセッサ上の資源に割り当てる仮想スレッド識別番号を適宜切り替えることによって、実際にハードウェア上に存在するスレッドより、仮想的に多くのスレッドが走行しているようにみせることができる。これによって、プログラマにハードウェアで用意されたより多くのスレッドを見せることが可能になり、プログラミングの自由度が増す。

【０１１１】また、本発明によれば、同期機構を実現するためのカウンタを設け、同期にかかわる複数のスレッドが同期命令によってこのカウンタをデクリメントすることによって、効率的にスレッド間の同期をとることができる。

【図面の簡単な説明】

【図１】本発明の一実施形態に係る仮想マルチスレッドプロセッサの構成例を示す図

【図２】同実施形態における仮想スレッドの状態およびその状態間の遷移関係について示す図

【図３】同実施形態におけるレーテンシの長い命令が実行される際の処理手順の一例を示すフローチャート

【図４】同実施形態におけるレーテンシの長いハードウェア資源への要求に対する応答を受けた場合の処理手順の一例を示すフローチャート

【図５】同実施形態におけるシステムタスクに関する処理手順の一例を示すフローチャート

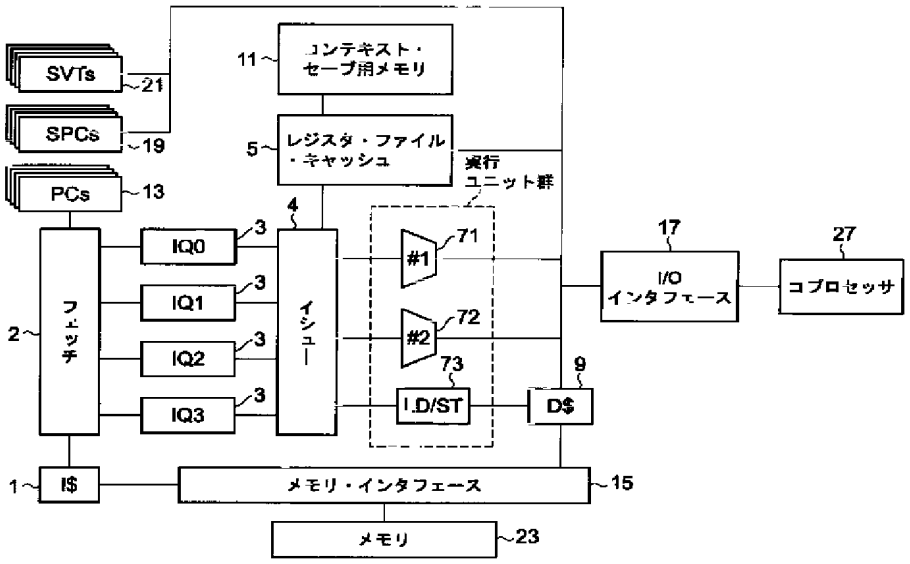
【図６】同実施形態におけるスレッド間の同期方法について説明するための図

【図７】同実施形態におけるスレッドがＳＹＮＣ命令に到達した場合の処理手順の一例を示すフローチャート

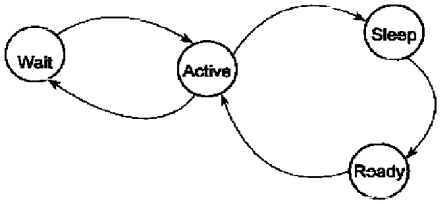
【符号の説明】

１…インストラクション・キャッシュ
２…フェッチ部
３…インストラクション・キュー
４…イシュー部
５…レジスタ・ファイル・キャッシュ
９…データ・キャッシュ
１１…コンテキスト・セーブ用メモリ
１３…プログラムカウンタ
１５…メモリ・インタフェース
１７…Ｉ／Ｏインタフェース
１９…ＳＰＣレジスタ
２１…ＳＶＴレジスタ
２３…メモリ
２７…コプロセッサ
７１，７２…演算器
７３…ロード・ストア・ユニット

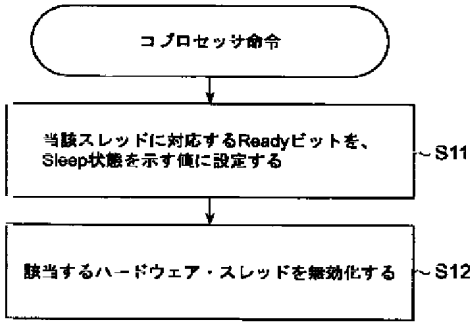
【図１】



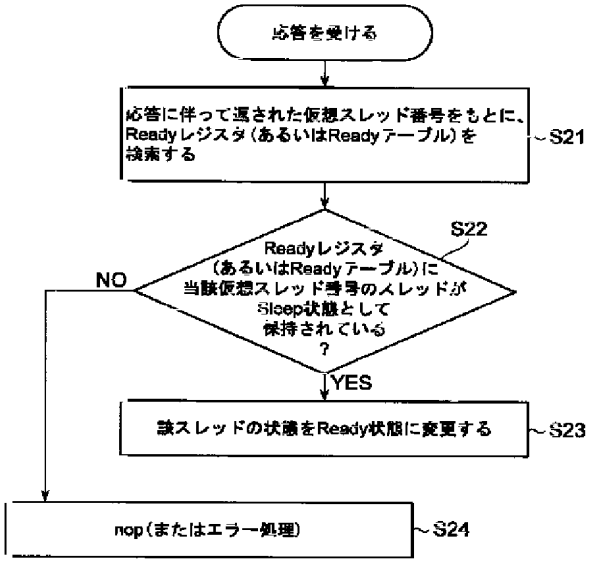
【図2】



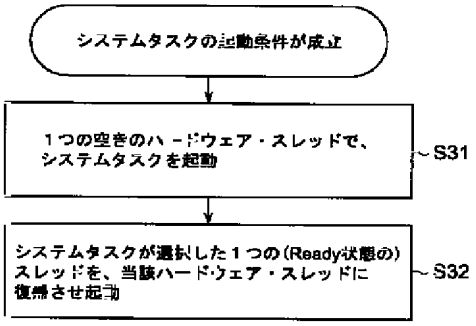
【図3】



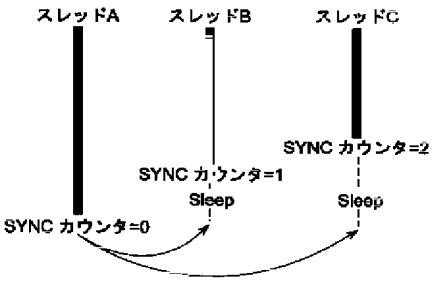
【図4】



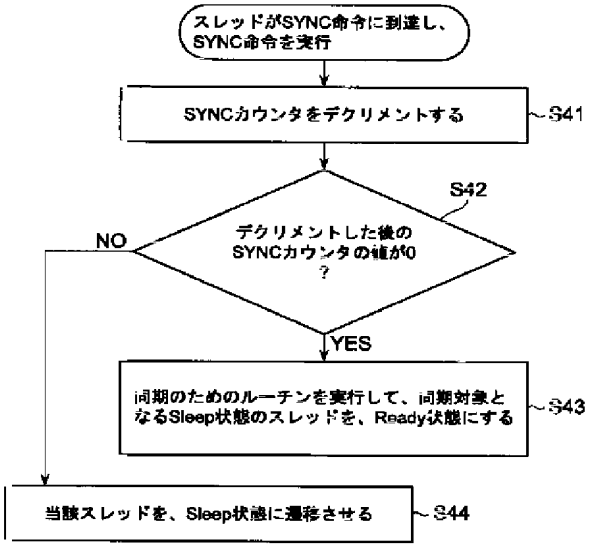
【図5】



【図6】



【図7】



フロントページの続き

Ｆターム(参考)　5B005 LL11 MM01
5B098 FF01 GA05 GC01 GD02 GD05
GD14